

# Symbolic Test generation using verification

Thierry Jéron  
Irisa/Inria Rennes, France  
<http://www.irisa.fr/vertecs/>

Adapted from

B. Jeannet, T. Jéron, V. Rusu, E. Zinovieva,  
**Symbolic Test Selection based on Approximate Analysis**,  
*in TACAS'05, LNCS 3440*, Edinburgh (Scotland), April 2005.

Vlad Rusu, Hervé Marchand, Thierry Jéron,  
**Automatic Verification and Conformance Testing for Validating Safety Properties  
of Reactive Systems**,  
*in Formal Methods 2005 (FM05)*, July 2005.

## Outline

---

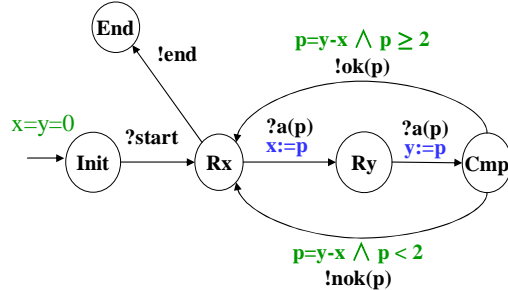
- The *ioSTS* model
  - Conformance Testing Theory with *ioco*
  - Test selection using approximate analysis
  - Conclusion
-

### 1. The *ioSTS* model

$\mathcal{S} = (V_S, \Theta_S, \Sigma, T_S)$  with

- $V_S$ : vector of variables  $\ni$  loc valuations  $v_i$  of  $v_i$  in  $\text{Dom}(v)$
- $\Theta_S$ : initial condition
- $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_c$ :  
 alphabet of actions  
 with comm. parameters  $p \in P$   
 $\text{sig}(a)$ : type of comm. param
- $T_S$ : transition relation

$[a(p) : G(v_S, p); v_S := A(v_S, p)]$   
 action guard assignment  
 also noted  $[a, p, G, A]$



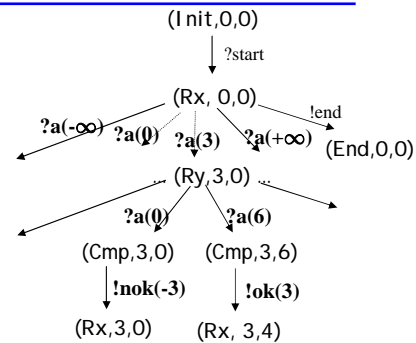
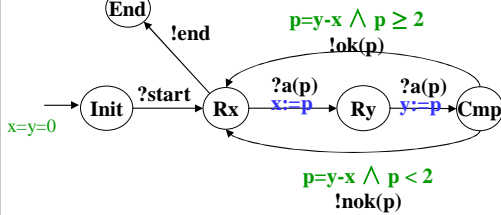
$V_S = \{\text{loc}, x, y\}$   $\text{Dom}(x) = \text{Dom}(y) = \mathbb{Z}$   
 $\Theta_S = \{x=y=0\}$   
 $\Sigma_1 = \{\text{!end}, \text{!ok}, \text{!nok}\}$   $\Sigma_2 = \{\text{?start}, \text{?a}\}$   
 $P = \{p\}$

- Hyp:
- satisfiability of guards is decidable.
  - $\Theta_S$  has a unique solution in  $\text{Dom}(V_S)$

3

### ioLTS semantics of *ioSTS*

$\mathcal{S} = (V_S, \Theta_S, \Sigma, T_S)$



$[[\mathcal{S}]] = \mathcal{S} = (Q, Q_0, \Lambda, \rightarrow)$  with

- $Q = \text{Dom}(V_S) = \times_{v \in V_S} \text{Dom}(v)$   
 i.e.  $q = \langle v_0, \dots, v_n \rangle$  is a vector of valuations
- $Q_0 = \{q_0\}$  where  $q_0 = \langle v_0, \dots, v_n \rangle$  is the unique solution to  $\Theta_S$
- $\Lambda = \{\langle a, \pi \rangle \mid a \in \Sigma \wedge \pi \in \text{Dom}(\text{sig}(a))\}$
- $\rightarrow$  is defined by:  $[a, p, G, A] \in T, q \in Q, \pi \in \text{Dom}(\text{sig}(a)), G(q, \pi)$   
 $q - a(\pi) \rightarrow q' = A(q, \pi)$

4

## Runs, traces

$S = [[\mathcal{S}]]$  is an ioLTS thus

$\text{Runs}(\mathcal{S}) = \text{Runs}(S): q_0 \rightarrow^{a_1(\pi_1)} q_1 \rightarrow^{a_2(\pi_2)} q_2 \dots \in Q_0 \cdot (\Lambda \cdot Q)^*$   
 represent executions from the initial state

$\text{Tr}(\mathcal{S}) = \text{Tr}(S) : \text{proj}_{\Lambda_{\text{vis}}}(\text{runs}(S))$   
 projection of runs on visible actions

$\text{STr}(\mathcal{S}) = \text{STr}(S) = \text{Tr}(\Delta(S))$

5

## Deterministic ioSTS

**Def:** an *ioSTS*  $\mathcal{S}$  is *deterministic* iff its semantics is  
 a deterministic ioLTS  $[[\mathcal{S}]]$  Undecidable problem

**Def:**  $\mathcal{S}$  is *syntactically deterministic* if  $\mathcal{S}$  has no internal action  
 and  $\forall$  action  $a \in \Sigma, \bigcap_{t=[a, p, G_t, A_t]} G_t = \emptyset$

**Prop:**  $\mathcal{S}$  is syntactically deterministic  $\Rightarrow [[\mathcal{S}]]$  is deterministic

**NB:** Improvement using over-approximate reachability analysis

Let  $\text{reach}^\alpha \supseteq \text{reach}$ , if  $\forall$  action  $a, \bigcap_{t=[a, p, G_t, A_t]} G_t \cap \text{reach}^\alpha = \emptyset$

then  $\mathcal{S}$  is deterministic

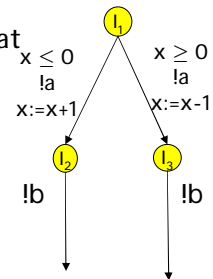
(e.g.  $\text{reach}^\alpha \subseteq I_1 \Rightarrow x \neq 0$ )

6

### Determinisation of *ioSTS*

**Problem:** given an *ioSTS*  $S$ , construct an *ioSTS*  $det(S)$  such that  $det(S)$  is deterministic and  $Tr(det(S)) = Tr(S)$

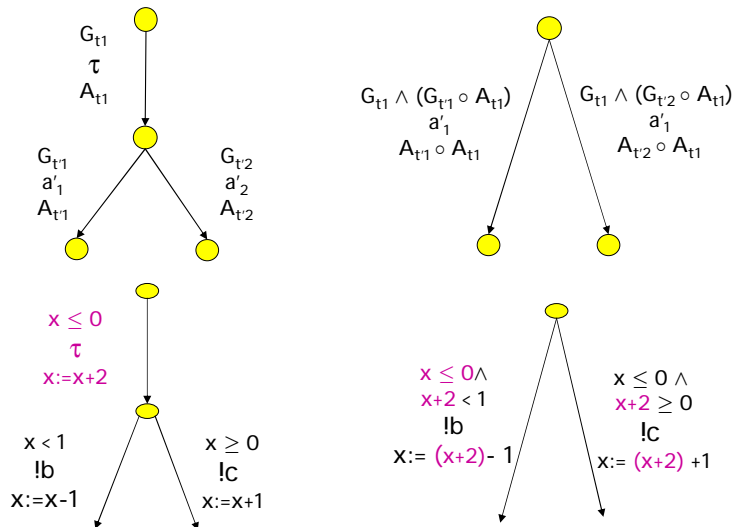
Determinisation of *ioSTS* into *ioSTS* is **not always possible** (deterministic *ioSTS* are a proper subclass of *ioSTS*)



- Syntactical heuristics with restrictions needed for termination:
- No internal loop for «  $\epsilon$ -closure »
  - Finite lookahead for « subset-construction »

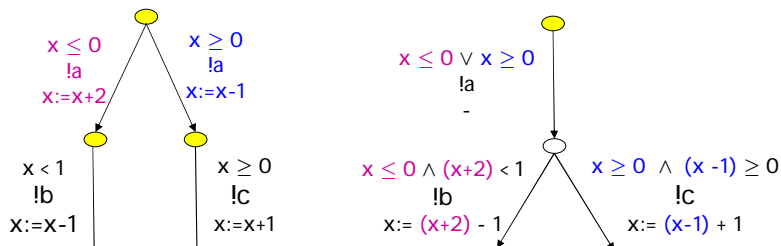
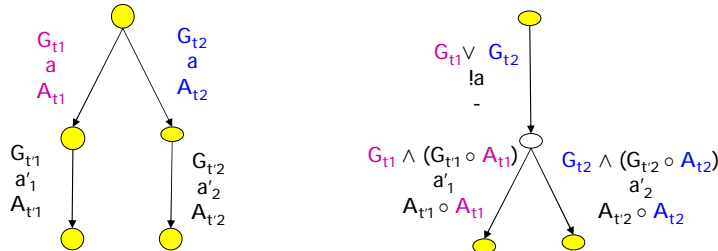
### Symbolic $\epsilon$ -closure

**Idea:** propagate assignments to next observable actions



### Determinisation heuristic

Idea: propagate assignments on next transitions until decision



9

### Quiescence

**Problem:** explicit quiescence by adding loops with ! $\delta$  in all quiescent states (no output is feasible)

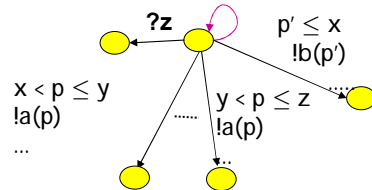
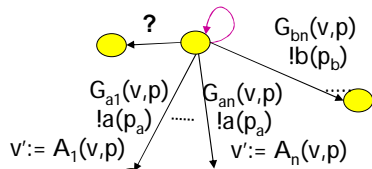
Transform  $\mathcal{S}$  syntactically in  $\Delta(\mathcal{S})$  such that  $[[\Delta(\mathcal{S})]] = \Delta([[S]])$

Augment guard model with universal quantification

not a real problem

$$\bigwedge_{a \in \Sigma_i \cup \Sigma_\tau} \neg (\bigvee_{t=[a,p,G,A]} \exists \pi, G(v,\pi) \quad !\delta)$$

$$\neg (\exists \pi, x < \pi \leq y \vee y < \pi \leq z) \wedge \neg (\exists \pi, \pi \leq x) \quad !\delta$$



10

### Simplyfying asumptions

---

- *ioSTS* are supposed to be deterministic
- Quiescence is not considered

---

11

### 2. Conformance Testing Theory with ioco [Tretmans 96]

---

- Specification: known ioLTS  $S$  (semantics of an ioSTS)
- Implementation: **unknown** ioLTS  $I$
  
- Conformance:  $I \text{ ioco } S$
  
- Test cases : ioSTS  $TC$  + Verdict variables
  - Execution: parallel composition  $\Delta(I) \parallel TC$
  - Verdicts:  $TC \text{ fails } I$
  
- Test generation:  $\text{gen\_test}: S \rightarrow TS = \{TC_1, TC_2, \dots\}$   
 Requested Properties of TS:  $TS \text{ fails } I \leftrightarrow \neg I \text{ ioco } S$   
 (soundness, limit exhaustiveness)

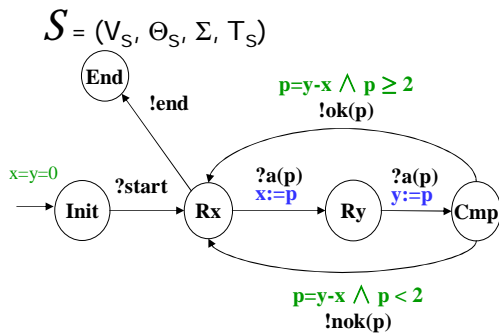
**Simplification: an automata/language point of view**

---

12

Conformance relation

**I ioco S ,  $\forall \sigma \in \text{STr}(S)$ ,  
 $\text{Out}(\Delta(I) \text{ after } \sigma) \subseteq \text{Out}(\Delta(S) \text{ after } \sigma)$**



Conformant traces, e.g.  
 ?start . ?a(4) . ?a(6) . !ok(2)  
 ?start . !end  
 ?start . ?start  
 (unspecified input allowed)

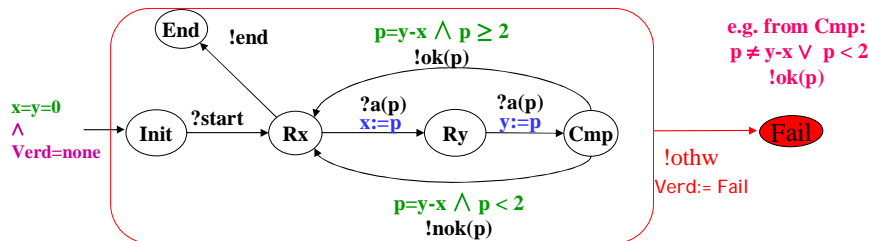
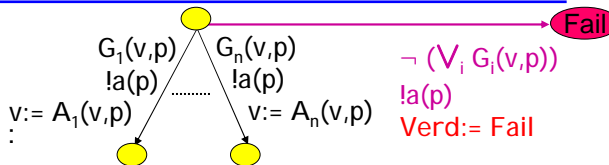
Non-conformant traces, e.g.  
 ?start . ?a(5) . ?a(7) . !ok(3)  
 ?start . ?a(6) . ?a(8) . !nok(2)

**Prop: I ioco S  $\Leftrightarrow \text{STr}(I) \cap [\text{STr}(S) . \Lambda_i^\delta \setminus \text{STr}(S)] = \emptyset$**   
 non-conformant behaviours

13

Canonical Tester  $\text{Can}(\mathcal{S})$ : observer of non-conformant behaviours

1. Add new variable Verd with initial value none  
 i.e.  $\Theta \wedge \text{Verd} = \text{none}$
2.  $\forall$  output !a,  $\forall t$  carrying !a :



**$\text{STr}_{\text{Fail}}(\text{Can}(\mathcal{S})) = \text{STr}(\mathcal{S}) . \Lambda_i^\delta \setminus \text{STr}(\mathcal{S})$**   
 $\Rightarrow$  I ioco S  $\Leftrightarrow \text{STr}(I) \cap \text{STr}_{\text{Fail}}(\text{Can}(\mathcal{S})) = \emptyset$

14

### Test cases, test execution, verdicts and properties

**Test Case:** deterministic ioSTS  $TC = (V_{TC}, \Theta_{TC}, \Sigma, T_{TC})$

+ verdict variables  $Verd \in \{\text{none}, \text{Fail}, \text{Pass}, \text{Inconc}, \dots\}$

plays the role of an observer delivering verdicts

$Tr_{Fail}(TC) = \{\sigma \in Tr(TC) \mid TC \text{ after } \sigma \in \text{Fail}\}$

**Test suite:** (infinite) set of test cases  $TS = \{TC_1, TC_2, \dots\}$

**Test execution:**  $TC \parallel \Delta(I)$  synchronization on common actions

**Possible rejection of I by TC:**

$TC \text{ fails } I, \quad STr(I) \cap Tr_{Fail}(TC) \neq \emptyset$

15

### Test suite properties

Possible rejection by a TC should correspond to non-conformance and vice-versa

$TC \text{ fails } I \Leftrightarrow STr(I) \cap Tr_{Fail}(TC) \neq \emptyset$

$I \text{ ioco } S \Leftrightarrow STr(I) \cap Tr_{Fail}(Can(S)) = \emptyset$

**TS is sound** ,  $\forall I, (I \text{ ioco } S \Rightarrow \forall TC \in TS, \neg TC \text{ fails } I)$

$\Leftrightarrow \bigcup_{TC \in TS} Tr_{Fail}(TC) \subseteq Tr_{Fail}(Can(S))$

**TS is exhaustive** ,  $\forall I, (\forall TC \in TS, \neg TC \text{ fails } I \Rightarrow I \text{ ioco } S)$

$\Leftrightarrow \bigcup_{TC \in TS} Tr_{Fail}(TC) \supseteq Tr_{Fail}(Can(S))$

16

### 3. Test selection for *ioSTS*

TS = {Can(S)} is a sound and exhaustive test suite but

- has too many (infinite) behaviours
- does not allow to control the implementation during testing

⇒ Test selection

- renounce to exhaustiveness in practice, select a finite TS likely to discover non-conformances
- focus on targetted behaviours of Can(S)
- use test purposes

17

### Test purposes

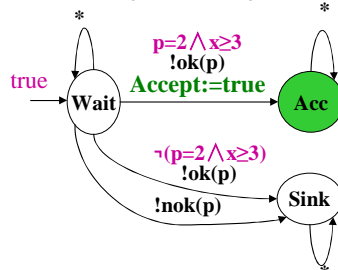
$$TP = (V_S \cup V_{TP}, \Theta_{TP}, \Sigma, T_{TP})$$

Observer of actions and variables of S

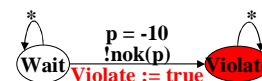
$$[a(p) : G(V_S, V_{TP}, p); V_{TP} := A(V_S, V_{TP}, p)] \in T_{TP}$$

Hyp : complete and deterministic

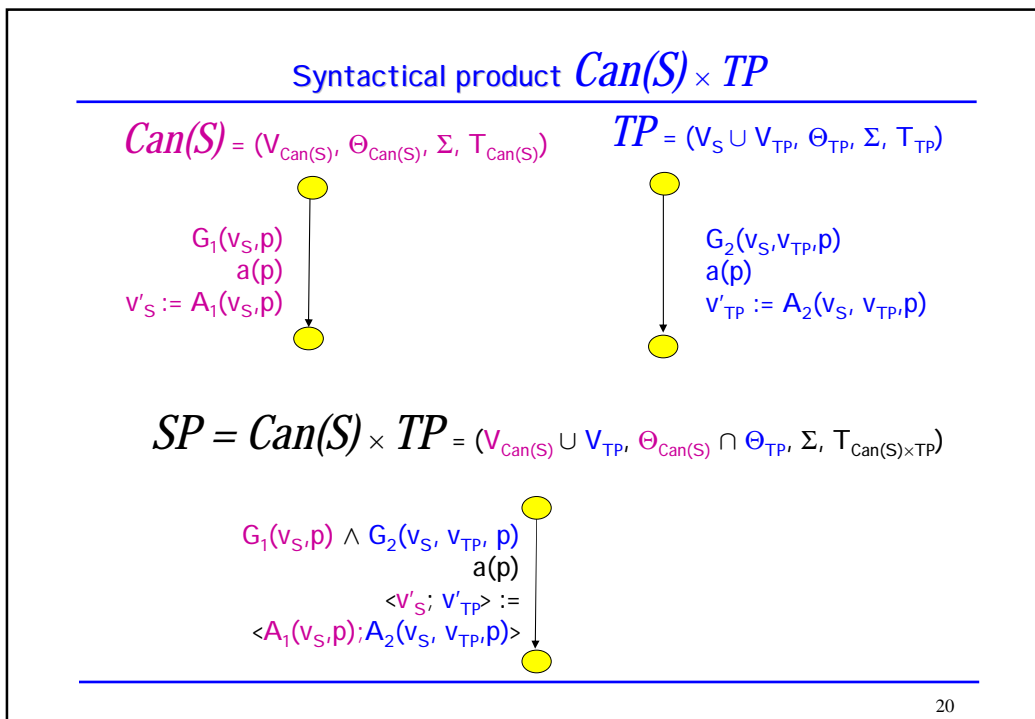
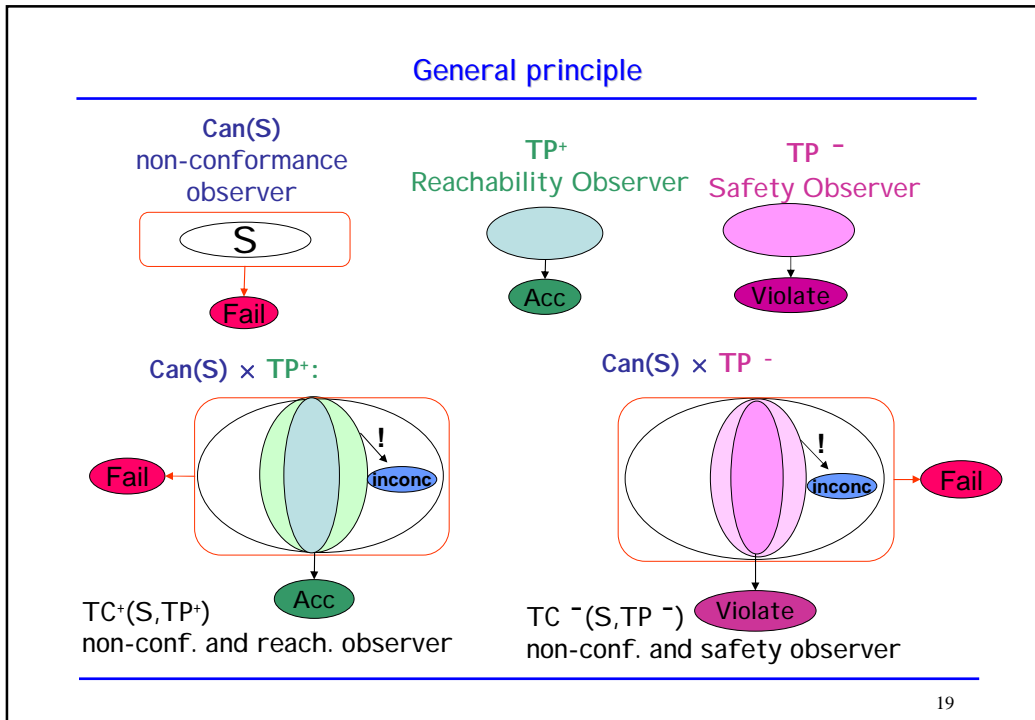
$TP+$ : reachability property

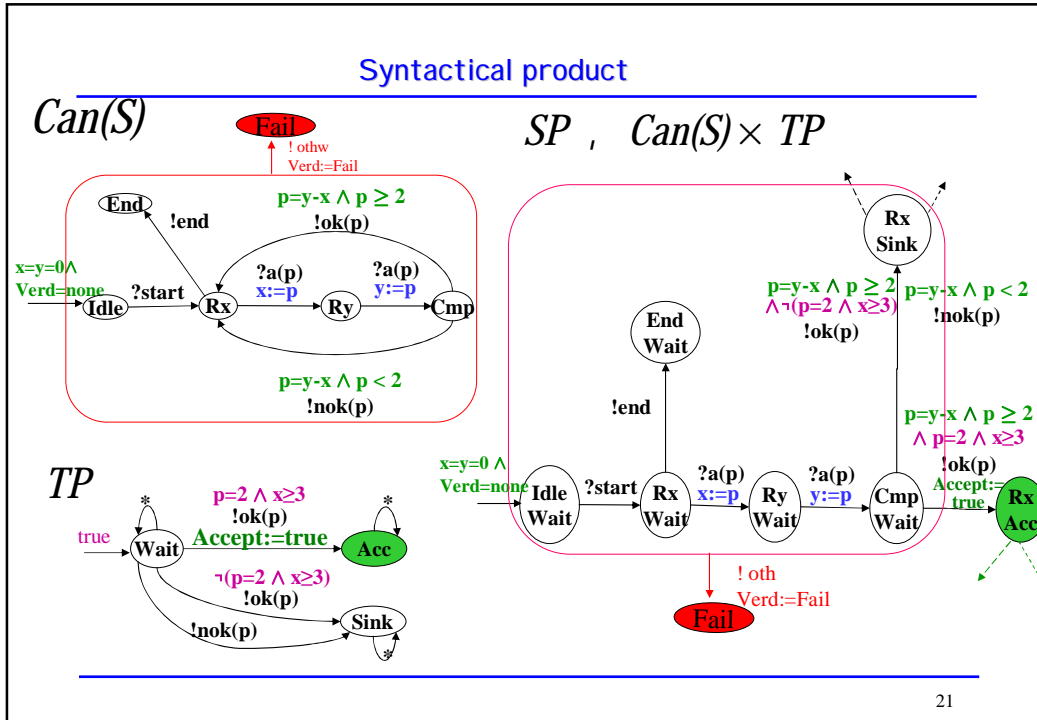


$TP-$ : negation of safety property



18





After product we get

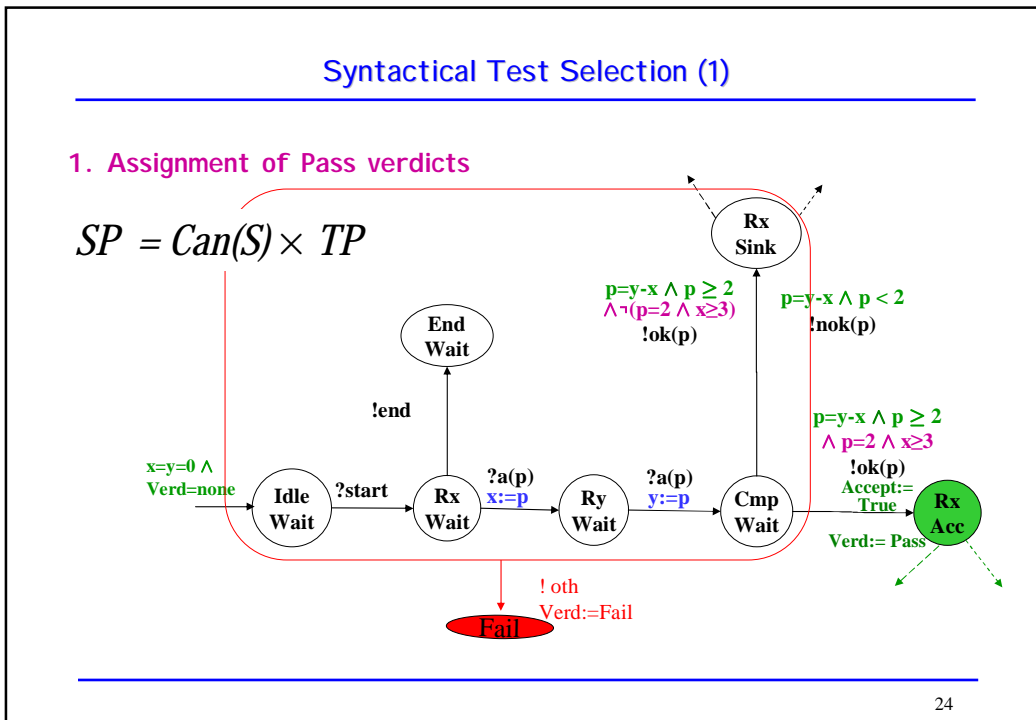
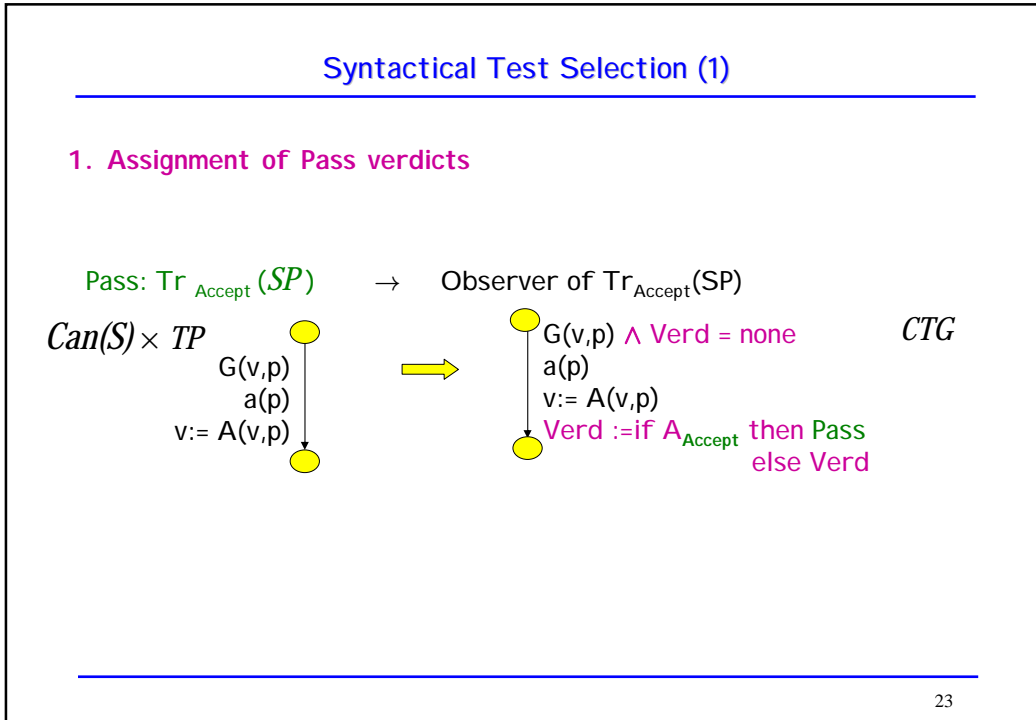
- $Tr(SP) = Tr(Can(S))$
- $Tr_{Fail}(SP) = Tr_{Fail}(Can(S))$
- $Tr_{Accept}(SP) = Tr_{Accept}(TP) \cap STr(S)$

SP is both a non-conformance and reachability observer  
but has too much behaviours:  $(Tr(Can(S)))$

**Goal of selection:**

- focus on  $Tr_{Accept}(TP) \cap STr(S)$ ,
- detect unfeasible traces to **Accept**
- Amounts to compute  $co\text{-}reach(Accept)$
- Undecidable  $\Rightarrow$  over-approximate

22

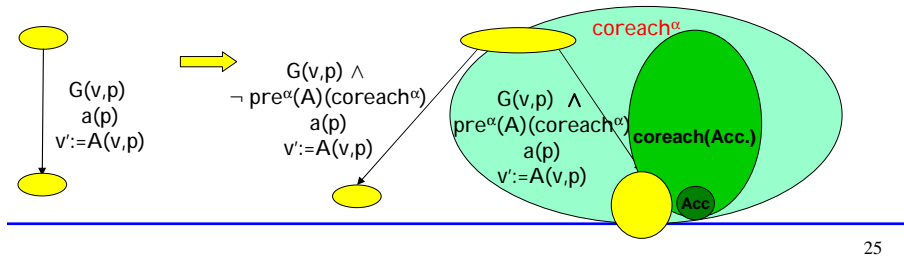


### Syntactical Test Selection (2)

#### 2. Selection and assignment of Inconc verdicts

coreach(Accept) not computable  $\Rightarrow$  compute over-approximation:  
 $\text{coreach}^\alpha \supseteq \text{coreach}(\text{Accept})$   
 $\forall \text{ assignment } A, \text{pre}^\alpha(A) (\text{coreach}^\alpha) \supseteq \text{pre}(A) (\text{coreach}^\alpha)$

**Idea:**  $\text{pre}^\alpha(A) (\text{coreach}^\alpha) = \text{Nec. Cond. to go into coreach}^\alpha$   
 $\neg \text{pre}^\alpha(A) (\text{coreach}^\alpha) = \text{Suf. Cond. to go outside coreach}^\alpha$   
 $\subseteq \text{outside coreach}(\text{Accept})$



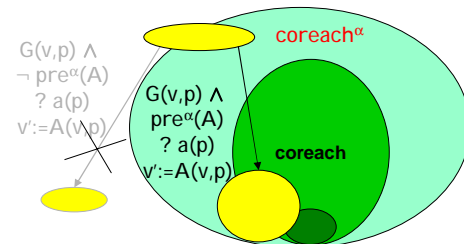
25

### Syntactical test selection (3): guard strengthening

#### Rule for inputs of $S$ :

keep conditions leading to  $\text{coreach}^\alpha$ ,

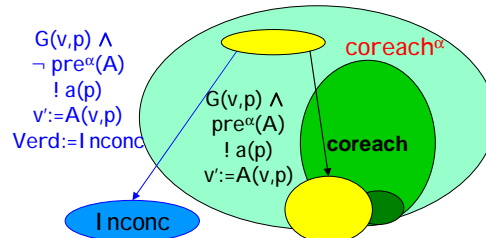
cut other ones (controllable):



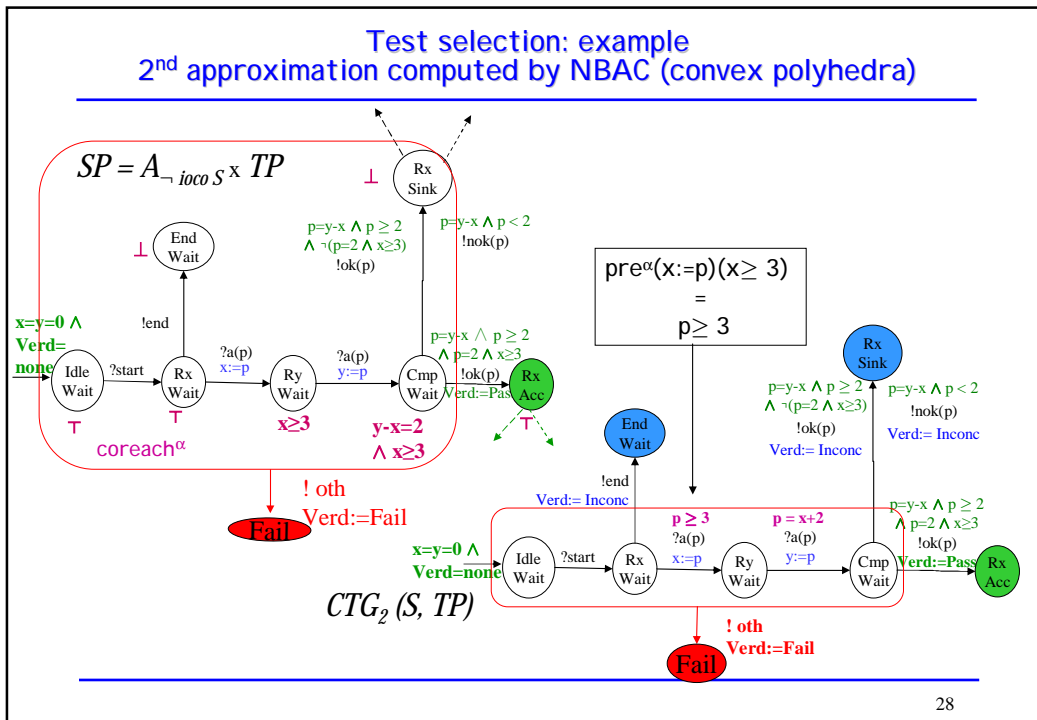
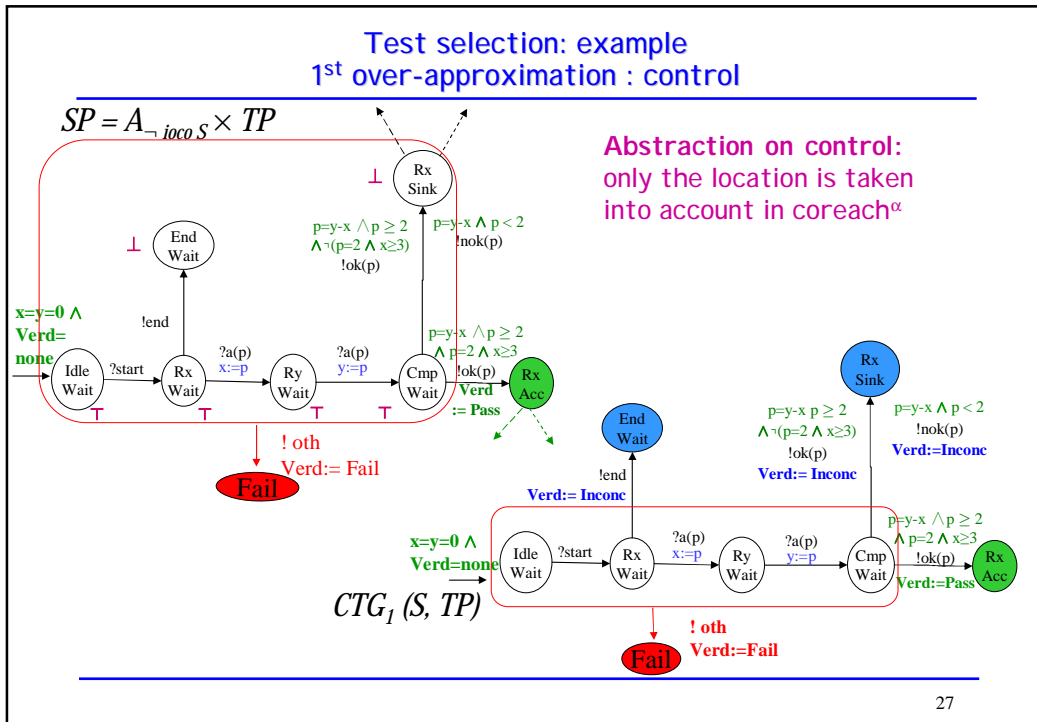
#### Rule for outputs of $S$

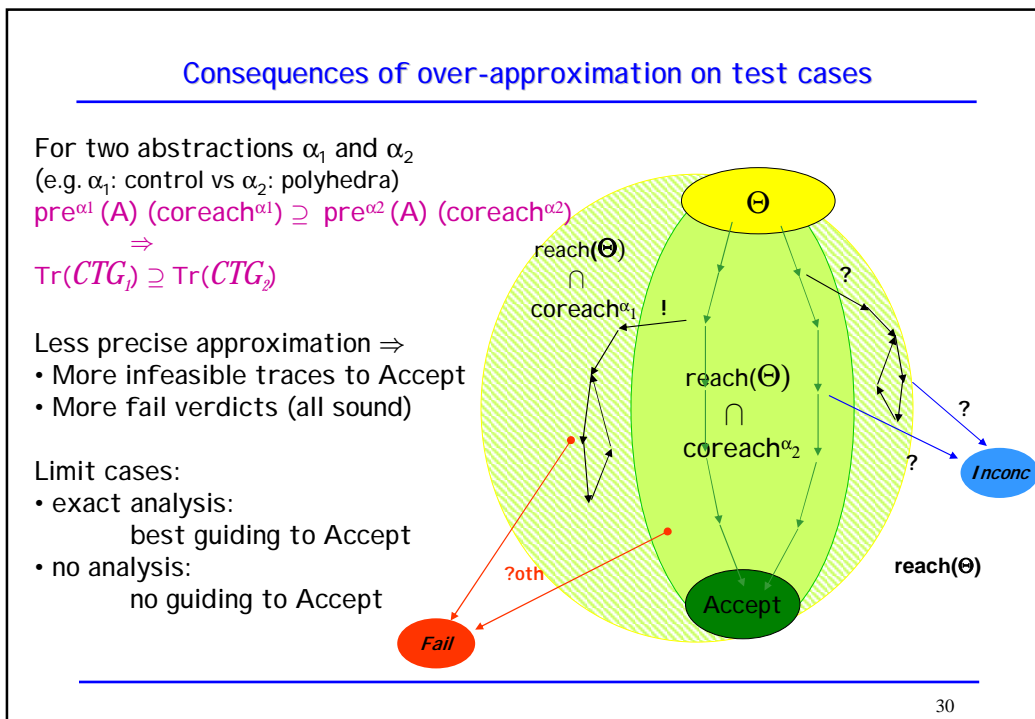
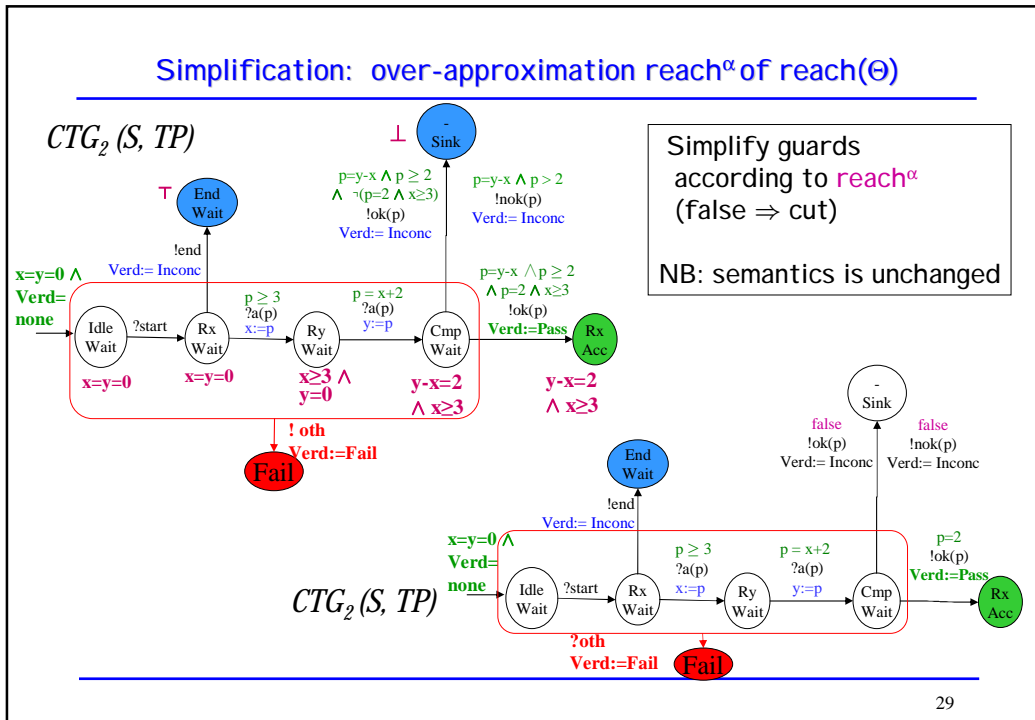
keep all conditions (uncontrollable),

those leading outside  $\text{coreach}^\alpha$   
 produce **Inconc**:



26





### Test execution

---

$CTG_2(S, TP)$

Inputs: [ ? a(p) : G(v,p); v:=A(v,p) ] : v is known,  
 choose  $\pi$  s.t. G(v, $\pi$ ), by **constraint solving**  
 send a( $\pi$ ), assign v:=A(v,  $\pi$ )

Outputs: [ ! a(p) : G(v,p); v:=A(v,p) ]: v is known,  
 receive a( $\pi$ )  
 evaluate G(v, $\pi$ )  
 if true, assign v:= A(v, $\pi$ ) (input complete)

?start . ?a(4) . ?a(6) . !ok(2) : **Pass**                      ?start . ?a(5) . ?a(7) . !ok(3) : **Fail**  
 ?start . !end : **Inconc**    ?start . ?a(6) . ?a(8) . !nok(2) : **Fail**

---

31

### Verification and Testing

---

---

32

### Model-checking a safety property

**S**

**TP** - =  $A_{\neg P}$

Model-checking  $S \models P$  ? reduces to reachability in  $S \times A_{\neg P}$  (undecidable)

$S \models \neg P$  : ?start.?a(10).?a(0).!nok(-10)  $\rightarrow$  (Rx, Violate)  
 With any abstraction  $S^\alpha \models \neg P$  ? is Yes  
 but  $S^\alpha \models \neg P$  ;  $S \models \neg P$   
 Result of  $S \models P$  ? could be Unknown

33

### Test selection from a safety observer

**Can(S)**

**$A_{\neg P} \text{ ioco } S \times A_{\neg P}$**

**$A_{\neg P}$**

34

## Some links between Model-checking and Conformance Testing

---

- Test selection using model-checking :
  - S deter., controllable, P reachability:  $TC \simeq \text{counter-exple of } S \stackrel{?}{=} P$   
[Engels et al. 97, Gargantini et al.99]
  - Extension to coverage using CTL [Hong et al.02] or observers [Blom et al.04]
  - Non-controllable case is more complex (this talk)
  
- Checking properties on the implementation
  - Black-box checking [Peled et al.] : learn I by experiment, model-check  $I \stackrel{?}{=} P$

---

 35

## Conclusion

---

Simplified and general framework for Ioco-based Test selection

- For finite ioLTS and infinite *ioSTS*
- Unified For **Reachability** and **Safety Observers**
- Using verification: coreachability analysis, over-approximations
- Completing verification (case of safety)

More research work needed for, e.g.

- Theories and algorithms for other models of reactive systems  
e.g. with time, data, stack, probabilities ...and combinations
- Coverage : measures, selection
- Links with structural testing techniques
- ....

---

 36